Master's Thesis

Deep Reinforcement Learning in Multi-End Games

Sol A Kim

Department of Computer Science and Engineering

Graduate School of UNIST

2020

Deep Reinforcement Learning in Multi-End Games

Sol A Kim

Department of Computer Science and Engineering

Graduate School of UNIST

Deep Reinforcement Learning in Multi-End Games

A thesis
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of Science

Sol A Kim

01.03.2020
Approved by

_____

Advisor
Kwang In Kim

Deep Reinforcement Learning in Multi-End Games

Sol A Kim

This certifies that the thesis of Sol A Kim is approved.

01.03.2020

_____

Advisor: Kwang In Kim

_____

Committee Member: Se Young Chun

_____

Committee Member: Jaesik Choi

Abstract

Recently deep reinforcement learning (DRL) algorithms show super human performances in the simulated game domains. In practical points, the sample efficiency is also one of the most important measures to determine the performance of a model. Especially for the environment of large search spaces (e.g. continuous action space), it is very critical condition to achieve the state-of-the-art performance. In this thesis, we design a model to be applicable to multi-end games in continuous space with high sample efficiency. A multi-end game has several sub-games which are independent each other but affect the result of the game by some rules of its domain. We verify the algorithm in the environment of simulated curling.

Contents

List of Figures

List of Tables

List of Abbreviations

CNN Convolutional Neural Network. 1, 2, 18

DRL Deep Reinforcement Learning. 1

HOO Hierarchical Optimistic Optimization. 3

KDE Kernel Density Estimation. 8, 18

KR Kernel Regression. 8, 18

KR-DL Kernel Regression-Deep Learning. 15, 17

KR-DL-UCT Kernel Regression-Deep Learning-Upper Confidence bound applied to Trees. 2, 4, 9, 15, 16

KR-DRL Kernel Regression-Deep Reinforcement Learning. 15, 17

KR-DRL-MES Kernel Regression-Deep Reinforcement Learning-Multi Ends Strategy. 15, 17

KR-UCT Kernel Regression-Upper Confidence bound applied to Trees. 3, 15

MCTS Monte Carlo Tree Search. 3, 5, 15, 18

MDP Markov Decision Process. 4

UCB Upper Confidence Bound. 6, 8, 9

I Introduction

Reinforcement learning is a large area of machine learning with supervised and unsupervied learning. With in an *environment* its goal is to train a controllable *agent* to take *optimal actions*. When the *agent* explores the *environment*, it gives feedbacks called *rewards* whether it goes right or wrong. The *optimal action* is an action maximizing the sum of future rewards.

Reinforcement learning framework is extensively applied to play diverse strategic games; chess [1], checkers [2], and othello [3]. Especially, deep reinforcement learning (DRL) which combines deep convolutional neural networks (CNNs) [4] to reinforcement learning methods have achieved super-human performance in Atari games [5] and Go [6, 7].

Through large deep learning networks, they compute probabilities for the most of scenarios and choose actions in discretized action space (e.g. Go has 19x19 possible actions per move). However, in continuous action space, we confront to the discretization problem to apply their frameworks. It is inevitable to loss information during the process. For some tasks which need very precise control, it is fatal to loss it.

We propose a policy search framework which solves these issues with an efficient continuous search algorithm on the top of action samples extracted by a deep CNN. The network still use the discretized state and action spaces, but we relieve the restriction problems of discretization through conducting a local search in a well-designed physical simulator with sampling stochastic and continuous actions. Through this method, we expect following two benefits. First, the advantage of expressiveness of deep neural networks, which learn the structure of global policy. Second, an efficient policy search in continuous spaces with the physical simulator enables to find the precise actions and controls.

We verify our method with the domain of curling which is one of Olympic sports. Its search space is huge enough to be viewed as a challenging game domain. Also, the uncertainty and precision on the ice sheet make it to be evaluated as the most challenging Olympic sport intellectually. The standard size of the ice sheet is 5.00m by 45.720m, and the diameter of a curling stone is about 30cm. To draw the stone to desired place, the player should precisely control the stone within 10-15cm. Even if the player can precisely control the power, angle, and turn of his or her moves, the uncertainty of slippery and melting ice delivers the stone to the other place. Thus, the degree of the risk also should be considered during the decision making process. Playing the game itself is very hard to decide which strategies to take and to consider because of those challenges. Moreover, there is one more important characteristic which makes the problem complicatedly.

A curling game has several ends which are sub-games. They are independent each others, but the result of an end determines the rule of the right next end. Maximizing the score of each ends does not guarantee the result of the game. The reward function of deep reinforcement learning (DRL) frameworks should consider this kind of rule of games to achieve the goal or to win the games. However, if we consider the only long term reward, it is hard to expect the

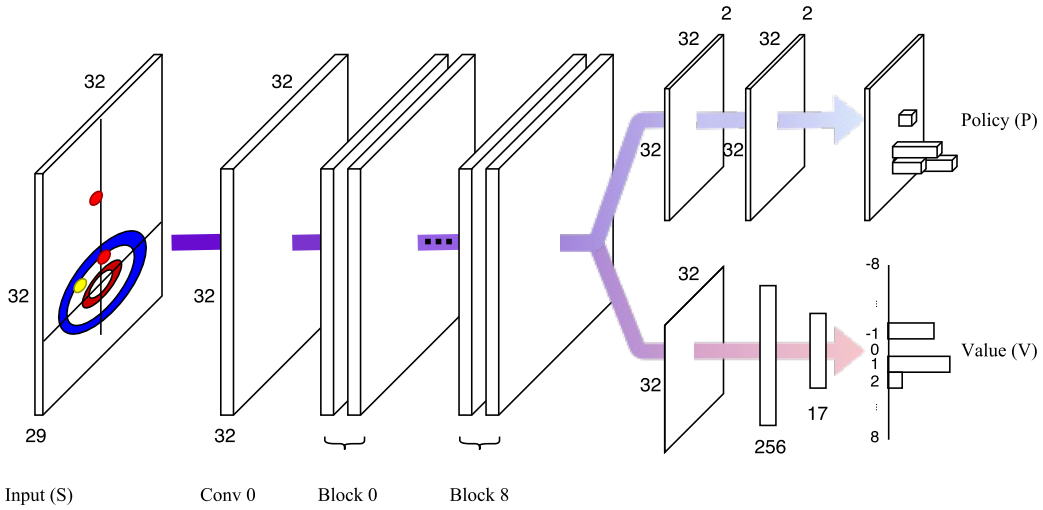

Figure 1: The architecture of our policy-value network. A feature map which indicates the state information is the input of this network. Each convolutional layer represents 32x32 discretized image of the position of stones. During the convolution this size is fixed without pooling. Policy and value network shares the bottom layers to increase the data efficiency. The probability distribution of actions is the output of policy head, and the score distribution is the output of value head. From the rule of curling, the maximum score which can be obtained by one end is eight. Thus its range is from -8 to 8.

convergency and high sample efficiency. We propose a method to handle the long term strategic game. First, we consider the short term reward per end. Then design an expected long term reward with a table constructed by statistical game results.

More specifically, we design a learning framework combines a new deep CNN which is called policy-value network and a stochastic continuous search method called kernel regression deep learning upper confidence bound applied to trees (KR-DL-UCT). The policy-value network gives the probability distribution of actions (i.e. policy network) and the score distribution which is considered as a expected reward (i.e. value network). We jointly trained the policy-value network as in Figure 1.

The program trained under our framework outperforms state-of-the-art digital curling programs, AyumuGAT'17 [8] and Jiritsukun'17 [9]. Our program also won in the Game AI Tournaments (GAT-2018) [10].

II Related work

In the domain of go, *AlphaGo Lee*, the successor version of *AlphaGo Fan* [6], defeated professional go players. Go has 19x19 discretized finite action space, although, the depth or length of play make complex branches. With the play data of human experts, the policy and value networks are trained in *AlphaGo Lee*. It also uses Monte Carlo tree search (MCTS) for policy improvement.

AlphaGo Zero [7] has 2,000 more elo rating score [11] than *AlphaGo Lee*, which is expected to win about 99.9% of matches against the latter one. It doesn't uses any hand-crafted features during the training. The learning starts from scratch, and the model is trained by itself through the self-play games. It also uses a unified policy and value networks to increase the data efficiency.

For the game of curling, several methods have been proposed. To deal with large continuous action space, search methods constructing game trees [9] have discretized the space and designed their own evaluation functions. To consider the given execution uncertainty, the value of action is calculated by averaging values of neighboring actions.

Monte carlo tree search (MCTS) [12–14] is a prevalent algorithm which has been used for game AI agents. [15,16] It is a simulation based tree search algorithm which selects and expends nodes of the tree to explore optimal moves. KR-UCT is a MCTS method, which uses pseudo-count approach [17] to handle large continuous search spaces. By estimating the information of a node of the tree using its neighbors', the number of simulations needed to find the optimal one is significantly lower than other tree search algorithms.

To deal with the continuous search space for the bandit problem, several algorithms have been used. Hierarchical optimistic optimization (HOO) [18] creates a cover tree, and divides the space recursively into small pieces of candidates for each depth. Each node of the cover tree is a arm of the sequential problem. During the recursion, it exploits the most promising one to create fine-grained estimates, and explores the region which has not been explored enough.

A well-designed simulator is very important to make decision for all search algorithms. Especially for the task applicable to real world problems, the similarity between the simulation and the reality affects reliability of the performance of algorithms. The game of curling corresponds to the task. Analyzing the dynamics of stones on the ice is important to design the accurate simulator. [19] However, it is not possible yet to modeling all changes of friction coefficients of the ice. Pebbles [20] is one of the main factors, which are the small granular frozen water on the ice sheet, make hard to estimate them between stones and the ice. Thus, general digital curling simulators assume a fixed friction coefficient which affects the result of simulation as an uncertainty of the execution. [21–23]

To implement and visualize the physical movement of curling stones, physics engines (e.g. Box2D [24], Unity3D [25] and Chipmunk 2D) are used. Parameters of the physics model are generated from the analysis of matching data between professional curling players [21,22,26]. In this paper, we use a curling simulator and its parameters from an international digital curling competition [21].

III Background

3.1 Reinforcemnt Learning

Markov Decision Process

Reinforcement learning problems can be modeled by markov decision process (MDP). With a fully observable environment, the model MDP has an omniscient viewpoint of given environment. Even if the environment is not perfectly observable, we can add some constraints to make the problem to a form of MDP.

When there is a change in the environment from a state s to the other state s', we define the work as an action a which is occurred during the transition. Problems of MDP assume only the state s and the action a affects to the transition to the state s'

MDP is a 5-tuple $< S, A, P, R, \gamma >$. S and A are finite sets of states s_t and actions a_t respectively for the time step $t = 0, 1, 2, \cdots, T$. P is a state transition probability matrix $P^a_{ss'} = \Pr[s_{t+1} = s' | s_t = s, a_t = a]$. R is a reward function of immediate or expected reward $R^a_s = \mathbb{E}[r_{t+1} | s_t = s, a_t = a]$ received by the environment after the transition, and γ is a discount factor which controls the importance of future rewards. The discount factor is a real number within $[0, 1]$. The lower value of discount factor represents the less importance of future rewards.

Policy Optimization

The purpose of reinforcement learning is to find the optimal policy which is the solution for a problem defined as MDP.

Policy π is a mapping function of actions from states to states in the set S. For the deterministic policy, the function is directly indicates an action a given a state s: $\pi(s) = a$. In the case of the stochastic policy, the function refers to the probability distribution of actions for given state: $\pi(a|s) = \Pr[a_t = a | s_t = s]$. We use the latter notation, because the former case can be represented by the one-hot vector which is composed by one '1' for the optimal action and '0's for the rest of actions.

The *optimal* policy is a policy maximizing expected discounted rewards

$$\mathbb{E}[\Sigma^{\infty}_{t=0} \gamma^t R^{a_t}_{s_t}]. \tag{1}$$

One method to find the *optimal* policy is dynamic programming with the recursive form of value function

$$v(s) = \mathbb{E}_{\pi}[R^{\pi(s_t)}_{s_t} + \gamma v(s_{t+1}) | s_t = s]. \tag{2}$$

The algorithm has two steps; a value update and a policy update. They are repeated in turn for all the states until no further changes observed. Both recursively update a new estimation of the optimal policy and value using an older estimation of those values.

Policy Iteration with Function Approximation

Policy iteration is an algorithm updates and generates the policy directly by iterating policy evaluation and improvement steps, rather than finds it indirectly by optimizing the value function. In large action space such as the continuous space and multi-dimensional spaces, function approximation make the use of realistic physical resource compare to create all possible cases with the form of table. Here we wrote the algorithm with an approximation function, deep neural network [27–29], which represent the policy and value functions through network parameters ρ and θ respectively.

Firstly, during the iteration, there is a step called policy evaluation to find a true value function. To judge the current policy π is a right one, a value function $v(s)$ is used for evaluating the importance of each state s through the policy π. The problem is the value function is not the real value function, so it should be updated through the iteration.

The parameterized value function v_θ is approximated by a neural network, so it is trained by the estimation. For given state s and reward $r(s)$, the estimator is updated using stochastic gradient descent method [30, 31],

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(r(s) - v_\theta(s)). \tag{3}$$

With the value function v_θ, the next step is to update the policy π_ρ. Through this procedure, the agent follows the better policy, and finally finds the optimal policy. For given state s and action a the policy can be trained by stochastic gradient ascent method to maximize the probability of action a is selected on state s,

$$\Delta\rho \propto \frac{\partial \log \pi_\rho(a|s)}{\partial\rho}. \tag{4}$$

Another method to update the policy is policy gradient reinforcement learning [32]. For each time step t, the policy π_ρ parameterized by ρ is trained by maximizing the expected result $r(s_t)$:

$$\Delta\rho \propto \frac{\partial \log \pi_\rho(a_t|s_t)}{\partial\rho} r(s_t). \tag{5}$$

3.2 Monte Carlo Tree Search

A Monte Carlo tree search (MCTS) [12–14] is a simulation based search algorithm. It is usually applied to games which needs controls and strategies to play. By constructing a game-tree for each decision making process, it samples an action to explore and then expands a node or visit any existing node for each iteration. Usually nodes and branches of the tree are equivalent to states and actions respectively, but it can be fluently changed by how implements the program. After expanding a new node, it simulates the game with a predifined *rollout function* (default policy) until the game is over to evaluate the play. At the terminal state, when its environment gives any return (e.g. score, win or lose), it evaluates the information of all the nodes of the path

from the leaf to root node, which called backpropagation. The information is usually composed of the number of visits corresponding node and a expected value of the node itself.

During the tree search algorithm, there is a important step to select which nodes to explore. One of the functions to choose them is an upper confidence bound (UCB) fucntion. The method which uses UCB as a selection function is called Upper confidence bounds applied to trees (UCT) [14].

More specifically, a node of the game-tree has two kinds of the information; the expected reward value $\bar{v}_a$ and the number of visits for corresponding action a. For each iteration of the algorithm, it visits an action which maximizing one side of a confidence interval Chernoff Hoeffding inequality [33] as,

$$\arg\max_a \bar{v}_a + C\sqrt{\frac{\log \sum_b n_b}{n_a}}. \tag{6}$$

It selects an action which has the maximum expected value $\bar{v}_a$ and the lowest visit number n_a at the same time. The constant C controls their importance to select the right action to explore.

3.3 Kernel Regression

With non-parametric techniques, we can design a function without any assumption of its shape or parameters. It is more general than the parametric methods, so we can estimate the distribution of data more accurately. Kernel regression is a non-parametric method to estimate the conditional expectation $E[y|x]$ of a random variable y given data x. One proposed function to estimate the expectation is averaging the variable y with a kernel function $K(x, x')$ which indicates the relation between two parameters x and x' [34–36],

$$\mathbb{E}[y|x] = \frac{\sum_{i=0}^{n} K(x, x_i) y_i}{\sum_{i=0}^{n} K(x, x_i)}. \tag{7}$$

Radial basis function kernel is a simple and popular kernel function defined as

$$K(x, x) = \exp\Big[\frac{||x - x'||^2}{2\sigma^2}\Big], \tag{8}$$

where σ is a free parameter. Intuitively we can interpret the meaning of the function as a similarity or distance between two points. The denominator of (7) is called kernel density estimation,

$$W(x) = \sum_{i=0}^{n} K(x, x_i), \tag{9}$$

which infers the density of a random variable based on given data.

3.4 The Game of Curling

Curling is also called chess on ice, which needs a great strategies to choose the most promising path to move and arrangement of stones for diverse situations. As the play is on the ice, there

is no absolutely prior strategy. The global goal of this game is to get the higher accumulated score than the opponent team's.

It is composed by even number of ends (8 or 10) except for the case of the overtime. In each end, eight stones are given to each team and the players throw their stones in turn on the ice sheet toward the circle called house (the scoring area). The score is determined right after the last (sixteenth) stone is thrown and stopped. A team which has a stone closer to the center of house called button than any other opponent's gets the score, and the number of stones corresponding to the rule is the amount of points they can get. In other words, the other team gets the point zero in the end.

The player who gets the point plays the next end as the first player. If there is no one who got the point, the turn of play is kept. The turn is very important to get the point, because a player who throw the last stone (hammer shot) can determine the result of each end. It indicates that the same strategy in one-end games makes different results in multi-end games. Thus, by optimizing the strategy for one-end games can not guarantee that it also can be a good strategy for multi-end games [23].

IV Deep Reinforcement Learning in Continuous Action Spaces

4.1 The Policy-Value Network

The advantages of the function approximation in reinforcement learning has been actively proved through performances of diverse games. [5–7]. In that sense, we design a similar architecture fo the domain of curling. After constructing features an input of 32x32 discretized image from the placement of stones, we stack convolutional layers and divide its head to represent the policy and value respectively. These approximated functions reduce the number of iterations to find the optimal action (policy network) and reduce the running time to simulate the scenarios to evaluate action explored (value network). The network architecture is described in Figure 1.

The policy network $\mathbf{p}_\theta$ has a network parameter θ, and outputs the probability distribution of actions which aim to 32x32x2 discretized board (32x32 grids for both clockwise and counter-clockwise spins).

The value network $\mathbf{v}_\theta$ shares its network parameter θ with the policy network to increase the data efficiency [6]. The value here indicates not a scalar number but a vector which is the probability distribution of scores $[-8, 8]$ which can be obtained in one end.

4.2 Search in Continuous Action Space

If the search space is very large or continuous, it is hard to get the global optimal action. When we choose the sampling method to search in continuous space, we can easily confront the localization problem. Even if using the random sampling, it needs huge number of samples to get the stable results, which is unpractical to play games in online matches. When we use the discretized continuous space, it would lead to get the global optimal action, but the loss of information during the discretization highly restricts its performance.

Our algorithm starts from discretized space with initialized actions using the policy network, but it is so meaningful that can solve the localization problem. Then, it estimate the values of each node based on the information of other nodes visited, using kernel methods (KR and KDE). With the estimators, it can update the information of each node indirectly. This procedure works as the information sharing method, which reduces the number of visits necessary to find the optimal action. With the UCT [14] framework, specifically, we use kernel based estimators. KR and KDE are used for estimating the expected value $\bar{v}_a$ and the number of visits n_a respectively.

$$\mathbb{E}(\bar{v}_a|a)=\frac{\sum\limits_{b\in A_t} K(a,b)\bar{v}_b n_b}{\sum\limits_{b\in A_t} K(a,b)n_b},\ W(a)=\sum_{b\in A_t} K(a,b)n_b \tag{10}$$

The method with those idea is applied to **Algorithm 1**, and its explanation is following as four steps of MCTS algorithm.

Selection First step is the selection. As a variation of UCB, we use kernel based estimators in line 9. Expected value and the visit number for each action are estimated by the information

Algorithm 1 KR-DL-UCT

1: $\mathbf{p}_\theta \leftarrow$ the policy network
2: $\mathbf{v}_\theta \leftarrow$ the value network
3: $s_t \leftarrow$ the current state
4: $A_t \leftarrow$ a set of visited actions in s_t
5: $expanded \leftarrow$ false
6: **if** s_t is terminal **then**
7: **return** Score(s_t), false
8: **end if**
9: $a_t \leftarrow arg\max_{a \in A_t} \mathbb{E}[\bar{v}_a|a] + C\sqrt{\frac{\log \Sigma_{b \in A_t} W(b)}{W(a)}}$
10: **if** $\sqrt{\sum_{a \in A_t} n_a} < |A_t|$ **then**
11: $s_{t+1} \leftarrow$ TakeAction(s_t, a_t)
12: $\vec{reward}, expanded \leftarrow$ KR-DL-UCT(s_t)
13: **end if**
14: **if not** *expanded* **then**
15: $a'_t \leftarrow arg\min_{K(a_t,a)>\gamma} W(a)$
16: $A_t \leftarrow A_t \cup a'_t$
17: $s_{t+1} \leftarrow$ TakeAction(s_t, a'_t)
18: $A_{t+1} \leftarrow \cup_{i=1}^{k} \{a_{init}^{(i)}\}$ s.t. $a_{init}^{(i)} \sim \pi_{a|s_{t+1}}$ // Policy net
19: $\vec{reward} \leftarrow \mathbf{v}_\theta(s_{t+1}|s_t, a'_t)$ // Value net
20: **end if**
21: $\vec{v}_{a_t} \leftarrow \frac{1}{n_{a_t}+1}(n_{a_t}\vec{v}_{a_t} + \vec{reward})$
22: $n_{a_t} \leftarrow n_{a_t} + 1$
23: **return** $\vec{reward}$, true

of sibling actions $b \in A_t$. To deal with one-end games, the value itself is used for calculate the expectation $\bar{v}_a$. For multi-end games, please refer Section VI. When the recursion reaches to the terminal (line 6), we select the action which has the number of most visits as the optimal action,

$$a^* = \arg\max_{a \in A_t} W(a), \tag{11}$$

because there is no reason to explore anymore for the last selection.

Expansion During the expansion, we use a technique called *progressive widening* to control the number of branches for each parent node. This indicates line 10. It makes the expansion can be occurred deeply rather than widely.

In this continuous search algorithm, the action which is selected by the UCB function is not the action to be explored or expanded directly. In line 15, it choose another action a'_t among the random sampled actions minimizing the estimated number of visits $W(a)$ and satisfying the similarity to the selected action a is greater than the constant τ. This is a method not only to

search in continuous space but also to explore efficiently.

In line 11 and 17, there is a function TakeAction(s_t, a_t). It is the simulation function to get the next state from the environment. That is, it returns the placement of stones after simulating chosen action and the statement of the game.

To utilize the kernel based estimators, it is necessary to define initial actions before estimating values. Also, it guides to start from the feasible region to find the optimal action. Our policy network π_a takes these role in line 18:

$$\pi_{a|s_{t+1}} = \frac{\mathbf{p}(a|s_{t+1})^{1/\tau}}{\Sigma_b \mathbf{p}(b|s_{t+1})^{1/\tau}}. \tag{12}$$

To sample the number of k actions for the initialization, we use a parameter tau which make the sampling can follow the distribution from the policy network.

Simulation For this step, we need a default policy called rollout to simulate scenarios until the terminal state to evaluate expanded nodes. Design of this function affects the performance of MCTS algorithms, because it directs to evaluate and update the parameters. The problem is that it is difficult to make the well-designed rollout at first. However, using our value network $\mathbf{v}_\theta$ which is trained based on the play data, we can directly get the value without constructing the default policy or even simulating the scenarios generated by the function. (line 19)

Backpropagation The last step is to update parameters of the nodes from the leaf to the root nodes along with the explored way of recursion. Variables along line 21 and 22 correspond to them.

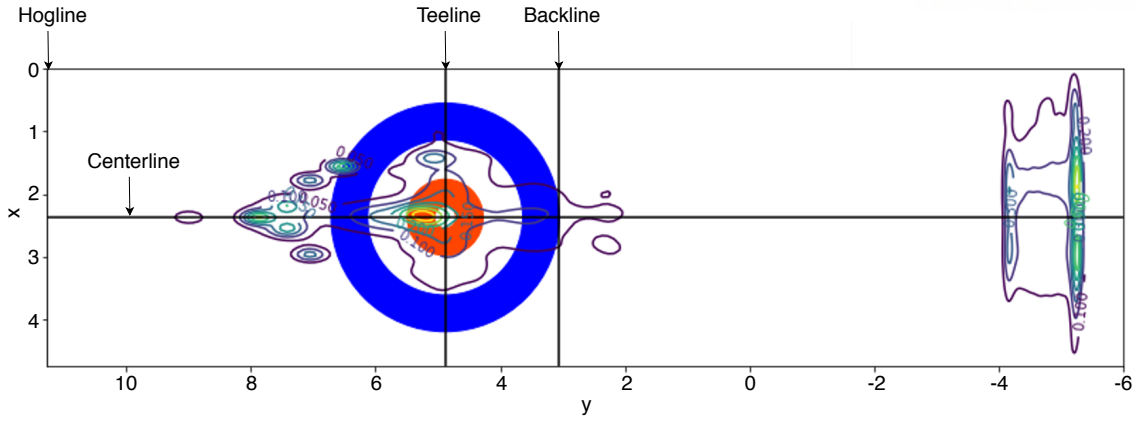

Figure 2: Kernel density estimation of shot data from the reference program for supervised learning. Shots whose spin is clockwise are analyzed only, and the opposite case is upside down with the centerline. During the simulation, a stone is thrown from the left to the right side. Stones stopped after the backline are ruled out. To takeout the opponent's stones, however, it is clear that the player aims to throw a stone behind the backline. For our reference program, the difference between shots whether to draw or to takeout the stones is obvious. Takeout shots are biased by [-6,-4] of y axis. Note that the curling sheet of this figure is not the state space (from hogline to backline) but the action space.

V Learning Pipeline

Our model is trained through two kinds of learning methods. One is *supervised learning* with data of a reference program, and the other is *reinforcement learning* with a pile of self-play data.

5.1 Supervised Learning

The policy-value network is first trained based on 4,000,000 shot data from the play of a reference program (Section 7.1). The data includes the information of the action (a), the placement of stones (s) and the scoreboard for each play. The policy network is trained by the pairs of action and state, and the value network estimate the possibility of getting each point for given state.

From a pile of reference data, m number of samples as a form of state and action pair (s_t, a_t). We make d-depth state s_{t+d} randomly by considering execution uncertainty. The state is evaluated by the value network and the prediction $\mathbf{z}_t$ is used for training the network.

To learn the designated policy $\boldsymbol{\pi}$ and value $\mathbf{z}$, the policy-value network $f_\theta(s)$ is trained to reduce the difference between the network pair $(\mathbf{p}_\theta, \mathbf{v}_\theta)$ and the data pairs $(\boldsymbol{\pi}, \mathbf{z})$.

We use stochastic gradient descent method to train the network. The sum of the cross-entropy losses for the training is defined as

$$l = -\mathbf{z}^T \log \mathbf{v}_\theta - \boldsymbol{\pi}^\mathbf{T} \log \mathbf{p}_\theta + c||\theta||^2, \tag{13}$$

where the constant c for the regularization parameter is set to 0.0001.

Here, we set $\pi_a = 1$ for the action a selected by the reference program, and $\pi_{b \in A \setminus \{a\}} = 0$ for others. We analyzed the reference program as in Figure 2 and we find that eliminating a part of the action space beyond the backline is very effective during the training as reduce the action space. Also it accelerate to learn strong shots (i.e. *takeout*[1] shots) as deleting the less important area. The network was trained for roughly 100 epochs. The learning rate was initialized by 0.01 and gradually reduced.

5.2 Self-Play Reinforcement Learning

We use policy iteration method to update the policy further through self-play reinforcement learning. The proposed actions are generated by **Algorithm** 1. That is, the data generated by the self-play games is used for train the policy-value network, and the network is also used for generate the data iteratively. Through the iterations, the policy and value form the network is gradually improved to find the true functions.

[1] An action to make a stone that hits another stone to remove it.

VI Learning Long Term Strategies

The number of	Difference of cumulative scores								
remaining ends	≤ -4	-3	-2	-1	0	1	2	3	≥ 4
10					0.418				
9					0.397	0.524	0.648	0.749	0.811
8	0.147	0.153	0.198	0.288	0.411	0.550	0.679	0.781	0.843
7	0.112	0.119	0.163	0.250	0.374	0.520	0.663	0.779	0.842
6	0.103	0.110	0.159	0.262	0.408	0.569	0.715	0.825	0.889
5	0.063	0.072	0.115	0.205	0.339	0.506	0.677	0.813	0.894
4	0.058	0.063	0.114	0.230	0.412	0.610	0.770	0.880	0.938
3	0.020	0.027	0.062	0.144	0.273	0.467	0.691	0.854	0.939
2	0.017	0.020	0.058	0.174	0.435	0.708	0.860	0.945	0.981
1	0.000	0.002	0.013	0.075	0.127	0.355	0.725	0.915	0.981
0	0.000	0.000	0.000	0.000	0.500	1.000	1.000	1.000	1.000

Table 1: Winning rates on the side of first player when the number of remaining ends and the difference of cumulative scores of two players are given. When the difference is zero after the last end, which means draw, the winning rate is considered as half.

6.1 Domain Knowledges of Curling

Curling is also called chess on ice, which needs a great strategies to choose the most promising path to move and arrangement of stones for diverse situations. As the play is on the ice, there is no absolutely prior strategy. The global goal of this game is to get the higher accumulated score than the opponent team's.

It is composed by even number of ends (8 or 10) except for the case of the overtime. In each end, eight stones are given to each team and the players throw their stones in turn on the ice sheet toward the circle called house (the scoring area). The score is determined right after the last (sixteenth) stone is thrown and stopped. A team which has a stone closer to the center of house called button than any other opponent's gets the score, and the number of stones corresponding to the rule is the amount of points they can get. In other words, the other team gets the point zero in the end.

The player who gets the point plays the next end as the first player. If there is no one who got the point, the turn of play is kept. The turn is very important to get the point, because a player who throw the last stone (hammer shot) can determine the result of each end. It indicates that the same strategy in one-end games makes different results in multi-end games. Thus, by optimizing the strategy for one-end games can not guarantee that it also can be a good strategy for multi-end games [23].

6.2 Multi-End Strategy

To consider the characteristics of multi-end games, we design a table (Table 1). Based on statistical result of the play data it converts the score to the winning rate with the information of remaining ends. In detail, the table uses two variables; the number of ends remained n and the difference of the sum of points δ. Following is the example of the conversion from a score to winning percentage.

Let's consider an situation for the case, the accumulated score is tied until the previous end, and a team which plays the first shot at the current end. If the team expected to get one point in this end, and there are only one ends which is remained. Then, the variables (n, δ) for Table 1 are set to (1, 1). That is the probability of the team wins for the game would be $P_{win}(n = 1, \delta = 1) = 35.5\%$.

In this way, the expected winning percentage $\bar{v}_a$ can be efficiently computed and used for replace the expected value in **Algorithm 1**. The score distribution *score* over the range of $[-8, 8]$ we can formalize the conversion to the expected winning rate,

$$\bar{v}_a = \mathbb{E}_v[P_{win}(n, \delta + score)]. \tag{14}$$

VII Experimental Results

To verify our algorithm, we use a simulator provided on the international simulated curling competition [21]. It includes the action uncertainty on the ice as an asymmetric Gaussian noise. Its visualization is implemented on Box2D physical engine, and the collision of stones is also observed.

7.1 Dataset

The play data used for training our policy-value network which is described in Section 5.1 is from running the top ranked program AyumuGAT'16 [8]. Its algorithm is also a MCTS algorithm and the winner of the Game AI Tournaments digital curling championship in 2016 (GAT-2016) [10].

7.2 Settings

We construct three models to verify our contributions. Firstly, a model named kernel regression-deep learning (KR-DL) is a program using the policy-value network trained based on the reference program only. (Section 5.1) The second program is kernel regression-deep reinforcement learning (KR-DRL) which trained further using the self-play data. (Section 5.2) The last program has overall ideas which are proposed in this paper, kernel regression-deep reinforcement learning-multi ends strategy (KR-DRL-MES). Note that our **Algorithm** 1 is used for all three programs.

For the configuration of **Algorithm** 1, we choose the constant values of free variables C and τ as 0.1 and 0.33 respectively through the analysis of the experimental results. During the self-play games the number of iterations of the algorithm is set to 400 which is enough to explore the search space to get the optimal strategy. It took a week to get 5,000,000 shot positions with 5 GPUs. During the self-play games our policy-network is trained using uniformly sampled data based on the most recent 1,000,000 shot data. The Table 1 is a statistical result based on the data for both supervised learning and self-play reinforcement learning.

7.3 Results

To verify our algorithms KR-DL-UCT itself, we make an experiment to match with the algorithms without the kernel based estimators, DL-UCT. As it shown in Figure 3 its performance already higher than DL-UCT at the start, before the policy-value network is trained further through the play. As the network is trained further, the difference between two program is increased until 16.05% by 5,000,000 shot data.

We set the baseline program as KR-UCT [22]. For each iteration, it uses 1,600 simulations to make decision, which is much higher than our configuration. It also uses hand-crafted features to generate rollout function.

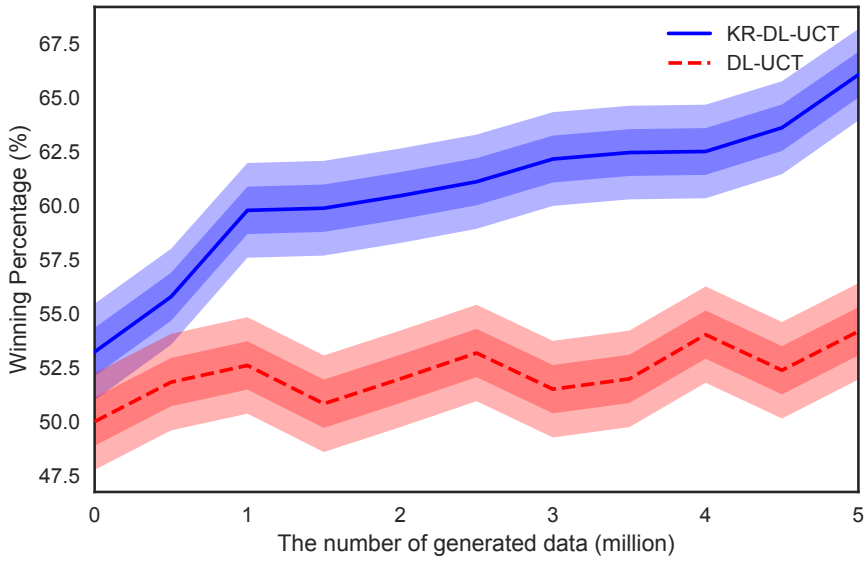

Figure 3: Learning curve for KR-DL-UCT and DL-UCT. The plot shows the winning percentages of 2,000 two-end games against DL-UCT with supervised learning only, 1,000 games as the first player and 1,000 games as the second player respectively. We compute the winning percentages by increasing the number of training shots.

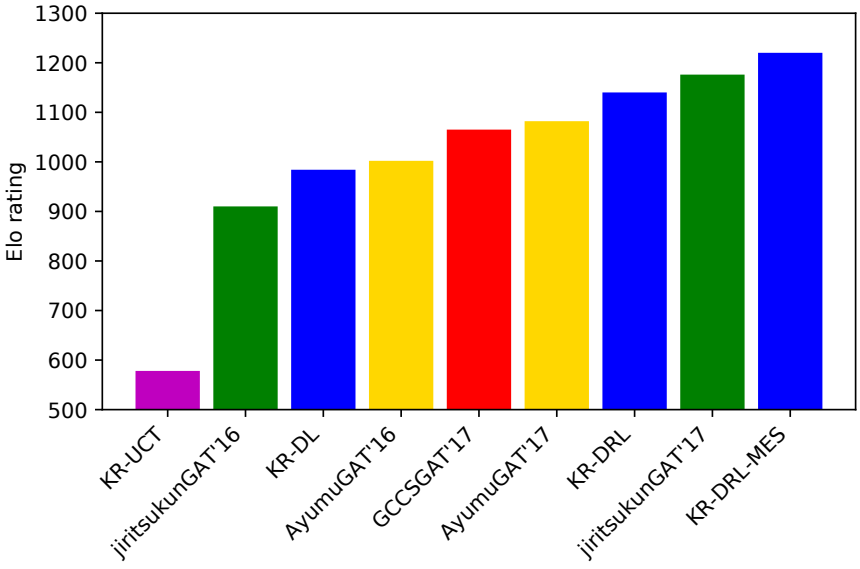

Figure 4: Elo rating and winning percentages of our models and GAT rankers. Each match has 200 games (each program plays 100 pre-ordered games), because the player which has the last shot (the hammer shot) in each end would have an advantage. Programs colored blue are our proposed programs.

The result of our three programs with the reference programs are shown in Figure 4. All games during the experiment follows the configurations of the latest GAT competition, which

allows only 3.4 seconds for thinking times per action, and each game is composed of eight ends. AyumuGAT'16 is the reference program for training the policy-value network in supervised learning manner, so our first program KR-DL performs a little bit worse than it. After training the network further based on the self-play data, KR-DRL outperforms it and other top ranked programs in the next year. JiritsukunGAT'17 is a program uses deep learning to identify the multi-end strategy and well-designed evaluation function to verify the value of each game state. It has the highest rating among the reference programs form the competition, but our program KR-DRL-MES outperforms it as in Figure 4.

VIII Conclusion

8.1 Summary

We propose a novel deep reinforcement learning method to handle the continuous search space and the multi-end game. Without designing hand-craft functions in the Monte Carlo tree search (MCTS) framework, we approximates the policy and value using the deep CNN. With the information sharing based on kernel regression (KR) and kernel density estimation (KDE), we release the restriction of using deep learning with the discretization. Also, we design the winning rate table to deal with the multi-end strategy and demonstrate that the algorithm equipped it shows the state-of-the-art performance in the domain of curling.

8.2 Future Work

Even if its statistical performance is good, the black box model makes hard itself to interact with human beings. The interaction encourage us to get the information and explain its behavior. This procedure increases the reliability of the performance and can be improved through feedbacks.

During the exploration and exploitation of the reinforcement algorithm, there are many expanded action which are not optimal at the end. It is clear that the number of experience of them is higher than any other selected actions if the algorithm works fine. We would like to analyze the reason why they are not selected to explain the reason why the optimal one is selected. We think this future work will be very interesting research topic to explain AI frameworks.

Acknowledgements

I would like to appreciate all people who have been supported me and my research. I'd like to express my gratitude to my advisor Jaesik Choi who is a mentor of all my work of the thesis. I would also like to thank all members of the Statistical Artificial Intelligence Laboratory. Each of them has been a help of my work not only as a coworker but as a friend. Thank you to my mother Jung Mi Ryu who always be my side and supports everything without any purpose. Finally, to all people who spend their time to help and support me, I would like to say thank you again.

References

[1] M. Campbell, A. Hoane Jr., and F.-H. Hsu, "Deep blue," *Artificial Intelligence*, pp. 57–83, 2002.

[2] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron, "A world championship caliber checkers program," *Artificial Intelligence*, pp. 273–289, 1992.

[3] M. Buro, "From simple features to sophisticated evaluation functions," in *Computers and Games*, 1999, pp. 126–145.

[4] Y. LeCun and Y. Bengio, "The handbook of brain theory and neural networks," 1998, pp. 255–258.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, pp. 529–533, 2015.

[6] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, pp. 484–489, 2016.

[7] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, pp. 354–359, 2017.

[8] K. Ohto and T. Tanaka, "A curling agent based on the monte-carlo tree search considering the similarity of the best action among similar states," in *Proceedings of Advances in Computer Games, ACG*, 2017, pp. 151–164.

[9] M. Yamamoto, S. Kato, and H. Iizuka, "Digital curling strategy based on game tree search," in *Proceedings of the IEEE Conference on Computational Intelligence and Games, CIG*, 2015, pp. 474–480.

[10] T. Ito, "The 4th uec-cup digital curling tournament in game artificial intelligence tournaments." [Online]. Available: http://minerva.cs.uec.ac.jp/curling/wiki.cgi?page=GAT_2018

[11] A. Elo, *The rating of chess players, past and present*, New York, 1978.

[12] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, pp. 1–43, 2012.

[13] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *Computers and Games*, 2007, pp. 72–83.

[14] L. Kocsis and c. Szepesvári, "Bandit based monte-carlo planning," in *Proceeding of European Conference on Machine Learning, ECML*, 2006, pp. 282–293.

[15] S. James, G. Konidaris, and B. Rosman, "An analysis of monte carlo tree search," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[16] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai." in *AIIDE*, 2008.

[17] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Proceedings of the Neural Information Processing Systems, NIPS*, 2016, pp. 1471–1479.

[18] S. Bubeck, G. Stoltz, C. Szepesvári, and R. Munos, "Online optimization in x-armed bandits," in *Proceedings of the Neural Information Processing Systems, NIPS*, 2008, pp. 201–208.

[19] E. Lozowski, K. Szilder, S. Maw, A. Morris, L. Poirier, and B. Kleiner, "Towards a first principles model of curling ice friction and curling stone dynamics," pp. 1730–1738, 2015.

[20] N. Maeno, "Dynamics and curl ratio of a curling stone," *Sports Engineering*, pp. 33–41, 2014.

[21] T. Ito and Y. Kitasei, "Proposal and implementation of digital curling," in *Proceedings of the IEEE Conference on Computational Intelligence and Games, CIG*, 2015, pp. 469–473.

[22] T. Yee, V. Lisý, and M. Bowling, "Monte carlo tree search in continuous action spaces with execution uncertainty," in *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, 2016, pp. 690–697.

[23] Z. Ahmad, R. Holte, and M. Bowling, "Action selection for hammer shots in curling," in *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, 2016, pp. 561–567.

[24] I. Parberry, *Introduction to Game Physics with Box2D*, 2013.

[25] S. Jackson, *Unity 3D UI Essentials*, 2015.

[26] M.-H. Heo and D. Kim, "The development of a curling simulation for performance improvement based on a physics engine," *Procedia Engineering*, pp. 385–390, 2013.

[27] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[28] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[29] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[30] J. Kiefer, J. Wolfowitz *et al.*, "Stochastic estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952.

[31] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[32] R. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proceedings of the Neural Information Processing Systems, NIPS*, 1999, pp. 1057–1063.

[33] W. Hoeffding, "Probability inequalities for sums of bounded random variables," in *The Collected Works of Wassily Hoeffding*. Springer, 1994, pp. 409–426.

[34] E. A. Nadaraya, "On estimating regression," *Theory of Probability & Its Applications*, vol. 9, no. 1, pp. 141–142, 1964.

[35] G. S. Watson, "Smooth regression analysis," *Sankhyā: The Indian Journal of Statistics, Series A*, pp. 359–372, 1964.

[36] H. J. Bierens, "The nadaraya-watson kernel regression function estimator," 1988.